Successful Application of Software Reliability Engineering for the NASA Space Shuttle

Ted Keller and Norman F. Schneidewind, Successful Application of Software Reliability Engineering for the NASA Space Shuttle, Software Reliability Engineering Case Studies, International Symposium on Software Reliability Engineering, November 3, Albuquerque, New Mexico, November 4, 1997, pp. 71-82.

Abstract

The Space Shuttle avionics software represents a successful integration of many of the computer industry's most advanced software engineering practices and approaches. Beginning in the late 1970's this software development and maintenance project has evolved one of the world's most mature software processes applying the principles of the highest levels of the Carnegie Mellon University Software Engineering Institute's Capability Maturity Model, Trusted Software Methodology, and ISO 9001 Standards. This software process, considered to be a "best practice" by many software industry organizations includes state-of-the-practice software reliability engineering methodologies. Life-critical Shuttle avionics software produced by this process is recognized to be among the highest quality and highest reliability software in operation in the world. This case study explores the successful use of extremely detailed fault and failure history, throughout the software life cycle, in the application of software reliability engineering techniques to gain insight into the flight-worthiness of the software and to suggest "where to look" for remaining defects. The role of software reliability models and failure prediction techniques is examined and explained to allow the use of these approaches on other software projects. One of the most important aspects of such an approach, "how to use and interpret the results" of the application of such techniques is addressed .

Keywords: Verification and validation, software reliability measurement and prediction, safety critical software, risk analysis.

Space Shuttle Flight Software Application

The *Space Shuttle* Primary Avionics Software Subsystem (PASS) represents a successful integration of many of the computer industry's most advanced software engineering practices and approaches. Beginning in the late 1970's this software development and maintenance project has evolved one of the world's most mature software processes applying the principles of the highest levels of the Software Engineering Institute's Capability Maturity Model and ISO 9001 Standards. This software process, considered to be a "best practice" by many software industry organizations includes state-of-the-practice software reliability engineering (SRE) methodologies. Life-critical PASS produced by this process is recognized to be among the highest quality and highest reliability software in operation in the world. Using this application, we show how SRE can be applied to: interpret software reliability predictions, support verification and validation of the software, assess the risk of deploying the software, predict the reliability of the software, develop test strategies to bring the software into conformance with reliability specifications, and make reliability decisions regarding deployment of the software.

Reliability predictions are currently used by Lockheed-Martin Space Information Systems to add confidence to established positions regarding low failure probabilities for the PASS that are based on formal software certification processes. It is the purpose of this case study to share the experience obtained from the use of SRE on this project, since this application is judged by the Lockheed-Martin team to be a successful attempt to apply SRE to this software. The SRE techniques and concepts employed by Lockheed-Martin should be of value for other software systems.

Interpretation of Software Reliability Predictions

Successful use of statistical modeling in predicting the reliability of a software system requires a thorough understanding of precisely how the resulting predictions are to be interpreted and applied [9]. The PASS (430 KLOC) is frequently modified, at the request of NASA, to add or change capabilities using a constantly improving process. Each of these successive PASS versions constitutes an upgrade to the preceding software version. Each new version of the PASS (designated as an Operational Increment, OI) contains software code which has been carried forward from each of the previous versions ("previous-version subset") as well as new code generated for that new version ("new-version subset").We have found that by applying a reliability model independently to the code subsets according to the following rules, we can obtain satisfactory composite predictions for the total version:

(1) all new code developed for a particular version uses the same development process.

(2) all code introduced for the first time for a particular version is considered to have the same life and operational execution history

(3) once new code is added to reflect new functionality in the PASS, this code is only changed thereafter to correct faults.

It is essential to recognize that this approach requires a very accurate code change history so that every failure can be uniquely attributed to the version in which the defective line(s) of code were first introduced. In this way it is possible to build a separate failure history for the new code in each release. To apply SRE to your software system you should consider breaking your systems and processes down into smaller elements to which a reliability model can be more accurately applied. Using this approach, we have been successful in applying SRE to predict the reliability of the PASS for NASA.

Estimating Execution Time

We estimate execution time of segments of the PASS software by analyzing records of test cases in digital simulations of operational flight scenarios as well as records of actual use in *Shuttle* operations. Test case executions are only counted as "operational execution time" for previous-version subsets of the version being tested if the simulation fidelity very closely matches actual operational conditions. Pre-release test execution time for the new code actually being tested in a version is never counted as operational execution time. We use the failure history and operational execution time history for the new-code subset of each version to generate an individual reliability prediction for that new code in each version by separate applications of the reliability model. This approach places every line of code in the total PASS into one of the subsets of "newly" developed code, whether "new" for the original version or any subsequent version. We then represent the total reliability of the entire software system as that of a composite system of separate components ("new-code subsets"), each having an individual execution history and reliability, connected in series. Lockheed-Martin is currently using this approach to apply the *Schneidewind* [8,9] model as a means of predicting a "conservative lower bound" for the PASS reliability.

Interpretations and Credibility

The two most critical factors in establishing credibility in software reliability predictions are the validation method and the way the predictions are interpreted. For example, a "conservative" prediction can be interpreted as providing an "additional margin of confidence" in the software reliability, if that predicted reliability already exceeds an established "acceptable level" or requirement. You may not be able to validate that you can predict the reliability of your software precisely, but you can demonstrate that with "high confidence" you can predict a lower bound on the reliability of that software within a specified environment. If you can use historical failure data at a series of previous dates (and you have the actual data for the failure history following those dates), you should be able to compare the predictions to the actual reliability and evaluate the performance of the model(s) used. You should take all these factors into consideration as you establish validation success criteria. This will also significantly enhance the credibility of your predictions among those who must make decisions based on your results.

Verification and Validation

Software reliability measurement and prediction are useful approaches to verify and validate software. Measurement refers to collecting and analyzing data about the observed reliability of software, for example the occurrence of failures during test. Prediction refers to using a model to forecast future software reliability, for example failure rate during operation. Measurement also provides the failure data that is used to estimate the parameters of reliability models (i.e., make the best fit of the model to the observed failure data). Once the parameters have been estimated, the model is used to predict the future reliability of the software. Verification ensures that the software product, as it exists in a given project phase, satisfies the conditions imposed in the preceding phase (e.g., reliability measurements of safety critical software components obtained during test conform to reliability specifications made during design) [5]. Validation ensures that the software product, as it exists in a given project phase, which could be the end of the project, satisfies requirements (e.g., software reliability predictions obtained during test correspond to the reliability specified in the requirements) [5].

Another way to interpret verification and validation is that it builds confidence that software is ready to be released for operational use. The release decision is crucial for systems in which software failures could endanger the safety of the mission and crew (i.e., safety critical software). To assist in making an informed decision, we integrate software risk analysis and reliability prediction, and we are evaluating stopping rules for testing. This approach is applicable to all safety critical software. Improvements in the reliability of software, where the reliability measurements and predictions are *directly related to mission and safety*, contribute to system safety.

Reliability Measurements and Predictions

There are a number of measurements and predictions that can be made of reliability to verify and validate the software. Among these are *remaining failures*, *maximum failures*, *total test time* required to attain a given *fraction of remaining failures*, and *time to next failure*. These have been shown to be useful measurements and predictions for: 1) providing confidence that the software has achieved reliability goals; 2) rationalizing how long to test a software component (e.g., testing sufficiently long to verify that the measured reliability conforms to design specifications); and 3) analyzing the risk of not achieving *remaining failure* and *time to next failure* goals [6]. Having predictions of the extent to which the software is not fault free (*remaining failures*) and whether a failure it is likely to occur during a mission (*time to next failure*) provide criteria for assessing the risk of deploying the software. Furthermore, *fraction of remaining failures* can be used as both an *operational quality* goal in predicting *total test time* requirements and, conversely, as an indicator of *operational quality* as a function of *total test time* expended [6].

The various software reliability measurements and predictions can be divided into the following two categories to use in combination to *assist* in assuring the desired level of reliability of the software in safety critical systems like the PASS. The two categories are: 1) measurements and predictions that are associated with residual software faults and failures, and 2) measurements and predictions that are associated with the ability of the software to complete a mission without experiencing a failure of a specified severity. In the first category are: *remaining failures*, *maximum failures*, *fraction of remaining failures*, and *total test time required to attain a given number or fraction of remaining failures*. In the second category are: *time to next failure* and *total test time required to attain a given time to next failure*. In addition, there is the risk associated with not attaining the required *remaining failures* and *time to next failure*. Lastly, there is *operational quality* that is derived from *fraction of remaining failures*. With this type of information a software manager can determine whether more testing is warranted or whether the software is sufficiently tested to allow its release or unrestricted use. These predictions provide a quantitative basis for achieving reliability goals [2].

Risk Assessment

Safety Risk pertains to executing the software of a safety critical system where there is the chance of injury (e.g., astronaut injury or fatality), damage (e.g., destruction of the *Shuttle*), or loss (e.g., loss of the mission) if a serious software failure occurs during a mission. In the case of the *Shuttle* PASS, where the occurrence of even trivial failures is extremely rare, the fraction of those failures that pose any safety or mission success impact is too small to be statistically significant. As a result, for this approach to be feasible, all failures (of any severity) over the entire 20-year life of the project have been included in the failure history database for this analysis. Therefore, the risk criterion metrics to be discussed for the *Shuttle* quantify the degree of risk associated with the occurrence of *any* software failure, no matter how insignificant it may be. The approach used can be applied to Safety Risk where sufficient data exist.

We are experimenting with an algorithm, which uses of the *Schneidewind Software Reliability Model* to compute a parameter: *fraction of remaining failures* as a function of the archived failure history during test and operation [6]. The prediction methodology uses this parameter and other reliability quantities to provide bounds on *total test time*, *remaining failures*, *operational quality*, and *time to next failure* that are necessary to meet arbitrarily defined *Shuttle* software reliability levels. The *total test time* versus *fraction of remaining failures* curve shows a pronounced asymptotic characteristic that indicates the possibility of big gains in reliability as testing continues; eventually the gains become marginal as testing continues.

Two criteria for software reliability levels will be defined. Then these criteria will be applied to the risk analysis of safety critical software, using the PASS as an example. In the case of the *Shuttle* example, the "risk" will represent the degree to which the occurrence of failures does not meet required reliability levels, regardless of how insignificant the failures may be. Next, a variety of prediction equations that are used in reliability prediction and risk analysis will be defined and derived; included is the relationship between *time to next failure* and *reduction in remaining failures*. Then it is shown how the prediction equations can be used to integrate testing with reliability and quality. An example is shown of how the risk analysis and reliability predictions can be used to make decisions about whether the software is *ready* to deploy; this approach could be used to determine whether a software system is *safe* to deploy.

Criteria for Reliability

If the reliability goal is the reduction of failures of a specified severity to an acceptable level of risk [7], then for software to be ready to deploy, after having been tested for total time t_t, it must satisfy the following criteria:

1) predicted remaining failures $r(t_t)<r_c$, (1)
where r_c is a specified critical value , and

2) predicted time to next failure $T_F(t_t)>t_m$, (2)
where t_m is mission duration. The total time t_t could represent a Safe/Unsafe criterion, or the time to remove all faults regardless of severity (as used in the *Shuttle* example).

For systems that are tested and operated continuously like the *Shuttle*, t_t, $T_F(t_t)$, and t_m are measured in execution time. Note that, as with any methodology for assuring software reliability, there is no guarantee that the expected level will be achieved. Rather, with these criteria, the objective is to reduce the risk of deploying the software to a "desired" level.

Remaining Failures Criterion

Using the assumption that the faults that cause failures are removed (this is the case for the *Shuttle*), *criterion 1* specifies that the residual failures and faults must be reduced to a level where the risk of operating the software is acceptable. As a practical matter, $r_c=1$ is suggested. That is, the goal is to reduce the expected *remaining failures* of a specified severity to less than one before deploying the software. The assumption for this choice is that one or more *remaining failures* would constitute an undesirable risk of failures of the specified severity. Thus, one way to specify r_c is by failure severity level (e.g., include) only life threatening failures). Another way, which imposes a more demanding criterion, is to specify that r_c represents *all* severity level, as in the *Shuttle* example. For example, $r(t_t)<1$ would mean that $r(t_t)$ must be less than one failure, *independent* of severity level.

If $r(t_t)\geq r_c$ is predicted, testing would continue for a total time $t_t'>t_t$ that is predicted to achieve $r(t_t')<r_c$, using the assumption that more failures will be experienced and more faults will be corrected so that the *remaining failures* will be reduced by the quantity $r(t_t)-r(t_t')$. If the developer does not have the resources to satisfy the criterion or is unable to satisfy the criterion through additional testing, the risk of deploying the software prematurely should be assessed (see the next section). It is known that it is impossible to demonstrate the absence of faults [3]; however, the risk of failures occurring can be reduced to an acceptable level, as represented by r_c. This scenario is shown in Figure 1. In *case A* $r(t_t)<r_c$ is predicted and the mission begins at t_t. In *case B* $r(t_t)\geq r_c$ is predicted and the mission would be postponed until the software is tested for total time t_t' when $r(t_t')<r_c$ is predicted. In both cases, *criterion 2)* would also be required for the mission to begin.

Time to Next Failure Criterion

Criterion 2 specifies that the software must survive for a time greater than the duration of the mission. If $T_F(t_t)\leq t_m$, is predicted, the software is tested for a total time $t_t''>t_t$ that is predicted to achieve $T_F(t_t'')>t_m$, using the assumption that more failures will be experienced and faults corrected so that the *time to next failure* will be increased by the quantity $T_F(t_t'')-T_F(t_t)$. Again, if it is infeasible for the developer to satisfy the criterion for lack of resources or failure to achieve test objectives, the risk of deploying the software prematurely should be assessed (see the next section). This scenario is shown in Figure 2. In *case A* $T_F(t_t)>t_m$ is predicted and the mission begins at t_t. In *case B* $T_F(t_t)\leq t_m$ is predicted and in this case the mission would be postponed until the software is tested for total time t_t'' when $T_F(t_t'')>t_m$ is predicted. In both cases *criterion 1)* would also be required for the mission to begin. If neither criterion is satisfied, the software is tested for a time which is the greater of t_t' or t_t''.

Total Test Time

The amount of *total test time* t_t can be considered a measure of the degree to which software reliability goals have been achieved. This is particularly the case for systems like the *Shuttle* where the software is subjected to continuous and rigorous testing for several years in multiple facilities, using a variety of operational and training scenarios (e.g., by Lockheed-Martin in Houston, by NASA in Houston for astronaut training, and by NASA at Cape Canaveral). We can view t_t as an input to a risk reduction process, and $r(t_t)$ and $T_F(t_t)$ as the outputs, with r_c and t_m as "risk criteria levels" of reliability that control the process. While it must be recognized that *total test time* is not the only consideration in developing test strategies and that there are other important factors, like the consequences for reliability and cost, in selecting test cases [11] nevertheless, for the foregoing reasons, *total test time* has been found to be strongly positively correlated with reliability growth for the *Shuttle* [9].

Remaining Failures

The mean value of the *risk criterion metric* (RCM) for *criterion 1* is formulated as follows:
RCM $r(t_t)= (r(t_t)-r_c)/r_c=(r(t_t)/r_c)-1$ (3)

Equation (3) is plotted in Figure 3 as a function of t_t for $r_c=1$, where *positive*, *zero*, and *negative* values correspond to $r(t_t)>r_c$, $r(t_t)=r_c$, and $r(t_t)<r_c$, respectively. In Figure 3, these values correspond to the following regions: *CRITICAL* (i.e., above the X-axis predicted *remaining failures* are greater than the specified value); *NEUTRAL* (i.e., on the X-axis predicted *remaining failures* are equal to the specified value); and *DESIRED* (i.e., below the X-axis predicted *remaining failures* are less than the specified value, which could represent a "safe" threshold or in the *Shuttle* example, an "error-free" condition boundary). This graph is for the *Shuttle Operational Increment OID* (with many years of shelf life): a software system comprised of modules and configured from a series of builds to meet *Shuttle* mission functional requirements. In this example, it can be seen that at approximately $t_t=57$ the risk transitions from the *CRITICAL* region to the *DESIRED* region.

Time to Next Failure

Similarly, the mean value of the *risk criterion metric* (RCM) for *criterion 2* is formulated as follows:

RCM $T_F(t_t)=(t_m-T_F(t_t))/t_m=1-(T_F(t_t))/t_m$ (4)

Equation (4) is plotted in Figure 4 as a function of t_t for $t_m=8$ days (a typical mission duration time for this OI), where *positive*, *zero*, and *negative* risk corresponds to $T_F(t_t)<t_m$, $T_F(t_t)=t_m$, and $T_F(t_t)>t_m$, respectively. In Figure 4, these values correspond to the following regions: *CRITICAL* (i.e., above the X-axis predicted *time to next failure* is less than the specified value); *NEUTRAL* (i.e., on the X-axis predicted *time to next failure* is equal to the specified value); and *DESIRED* (i.e., below the X-axis predicted *time to next failure* is greater than the specified value). This graph is for the *Shuttle operational increment OIC*. In this example the RCM is in the *DESIRED* region at all values of t_t.

Approach to Prediction

In order to support the reliability goal and to assess the risk of deploying the software, various reliability and quality predictions are made during the test phase to validate that the software meets requirements. For example, suppose the software reliability requirements state the following: 1) *ideally*, after testing the software for *total test time* t_t, the predicted *remaining failures* shall be less than one; 2) if the ideal of 1) cannot be achieved due to cost and schedule constraints, *time to next failure*, predicted after testing for *total test time* t_t, shall exceed the mission duration; and 3) the risk of not meeting 1) and 2) shall be assessed. Thus, this approach uses a software reliability model to predict the following: 1) *maximum failures*, *remaining failures*, and *operational* quality (as defined in the next section); 2) *time to next failure* (beyond the last observed failure); 3) *total test time* necessary to achieve required levels of *remaining failures* (fault) level, *operational quality*, and *time to next failure;* and 4) tradeoffs between increases in levels of reliability and quality with increases in testing (i.e., cost of testing).

An important concept to note is that reliability will be measured during test; that is, failure data will be collected for two purposes: 1) to verify that the observed data conform to the reliability specified during design and 2) to provide data for reliability parameter estimation. With regard to 1), the observed *time to next failure* can be compared to the specified quantity. However, in contrast, observed *remaining failures* and *maximum failures* have no meaning because we don't know how many *remaining failures* (faults) there are at a given time during the life of the software and we don't know the *maximum failures* that will have occurred at the end of the life of the software. Thus *remaining failures* and *maximum failures* only have meaning as predicted quantities. However, we can make approximations to these quantities for model validation purposes (see the *Summary of Predictions* section).

Prediction Equations

In order to consider the risk of deploying the PASS, various predictions have been made based on the *Schneidewind Software Reliability Model* [1, 8, 9, 10], one of the four models recommended in the *ANSI/AIAA Recommended Practice for Software Reliability* [1]. The equations are derived in the next section. They have been applied to analyze the reliability of the PASS based on the approach recommended herein. The *Statistical Modeling and Estimation of Reliability Functions for Software* (SMERFS) [4] is used for all predictions except t_t, which is not implemented in SMERFS.

Because the PASS is run continuously, around the clock, in simulation, test, or flight, "time" refers to continuous execution time and *total test time* refers to execution time that is used for testing. Failure count intervals are equal length periods of continuous execution time.

In the following equations, parameter a is the failure rate at the beginning of interval s; parameter ß is the negative of the derivative of failure rate divided by failure rate (i.e., relative failure rate); t is the last interval of observed failure data; s is the starting interval for using observed failure data in parameter estimation that provides the best estimates of a and ß and the most accurate predictions [8]; X_{s-1} is the observed failure count in the range [1,s-1]; $X_{s,t}$ is the observed failure count in the range [s,t]; and $X_t=X_{s-1}+X_{s,t}$. Failures are counted against *operational increments* (OIs). Data from four *Shuttle* OI's, designated *OIA, OIB, OIC,* and *OID* are used in this analysis example.

Cumulative Failures

When estimates are obtained for the parameters a and ß, with s as the starting interval for using observed failure data, the predicted *failure count in the range [s,t]* is obtained:

$F_{s,t}=(a/ß)[1-\exp(-ß((t-s+1)))]$ (5)

Furthermore, if X_{s-1}, the observed failure count in the range [1,s-1], is added to equation (5), the predicted *failure count in the range [1, t]* is obtained:

$F(t)=(a/ß)[1-\exp(-ß((t-s+1)))]+X_{s-1}$ (6)

Failures in an Interval Range

Let $t=t_2$ and subtract $X_{t1}=X_{s-1}+X_{s,t1}$, the observed failure count in the range $[1,t_1]$, from equation (6), then obtain the predicted *failure count in the range* $[t_1,t_2]$:

$F(t_1,t_2)=(a/ß)[1-\exp(-ß((t_2-s+1)))]-X_{s,t1}$ (7)

Maximum Failures

Let t=∞ in equation (6) and obtain the predicted *failure count in the range [1,¥]* (i.e., *maximum failures* over the life of the software):

$F(\infty)=a/ß+X_{s-1}$ (8)

Remaining Failures

To obtain predicted *remaining failures* r(t) at time t, subtract $X_t=X_{s-1}+X_{s,t}$ from equation (8):

$r(t)=(a/ß)-X_{s,t}=F(\infty)-X_t$ (9)

r(t) can also be expressed as a function of *total test time* t_t by substituting equation (5) for $X_{s,t}$ in equation (9) and letting $t=t_t$:

$r(t_t)=(a/ß)(\exp-ß[t_t-(s-1)])$ (10)

Fraction of Remaining Failures

If equation (9) is divided by equation (8), *fraction of remaining failures*, predicted at time t is obtained:

$p(t)=r(t)/F(\infty)$ (11)

Operational Quality

The *operational quality* of software is the complement of p(t). It is the degree to which software is free of remaining faults (failures), using the assumption that the faults that cause failures are removed. It is predicted at time t as follows:

$$Q(t)=1-p(t) \quad (12)$$

Total Test Time to Achieve Specified Remaining Failures

The predicted *total test time* required to achieve a specified *number of remaining failures* at t_t, $r(t_t)$,

$$t_t=[\log[a/(b[r(t_t)])]]/b+(s-1) \quad (13)$$

is obtained from equation (10) by solving for t_t:

Time to Next Failure

By substituting $t_2=t+T_F(t)$ in equation (7), letting $t_1=t$, defining $F_t=F(t,t+T_F)$,and solving for $T_F(t)$, the

$$T_F(t)=[(\log[a/(a-b(X_{s,t}+F_t)])/b]-(t-s+1)$$

$$\text{for } (a/b)>(X_{s,t}+F_t) \quad (14)$$

predicted *time for the next F_t failures* to occur, when the current time is t, is obtained :

The terms in $T_F(t)$ have the following definitions:

t:	Current interval;
$X_{s,t}$:	Observed failure count in the range [s,t]; and
F_t:	Given number of failures to occur after interval t.

Equations (5)-(11) and (14) are predictors of reliability that can be related to safety or, as in the *Shuttle* example, the error-free condition of the software; equation (13) represents the predicted *total test time* required to achieve stated reliability goals. If a quality requirement is stated in terms of *fraction of remaining failures*, the definition of *Q* as *Operational Quality*, equation (12), is the *degree* to which the software meets specified requirements [7]. For example, if a reliability specification requires that software is to have no more that 5% remaining failures (i.e., p=.05, Q=.95) after testing for a total of t_t intervals, then a predicted Q of .90 would indicate the *degree* to which the software meets the requirement.

Relating Testing to Reliability and Quality

Predicting Total Test Time and Remaining Failures

The tradeoff between testing and reliability can be analyzed by first using equation (8) to predict *maximum failures* ($F(\infty)=11.76$ for *Shuttle OIA*). Then, using given values of *p* and equation (11) and letting $t=t_t$, $r(t_t)$ is predicted for each value of *p*. The values of $r(t_t)$ are the predictions of *remaining failures* after the OI has been executed for *total test time* t_t. Next, the values of $r(t_t)$ and equation (13) are used to predict corresponding values of t_t. The results are shown in Figure 5, where $r(t_t)$ and t_t are plotted against *p* for *OIA*. Note that required *total test time* t_t rises very rapidly at small values of *p* and $r(t_t)$. Also note that the maximum value of *p* on the plot corresponds to $t_t=18$ and that smaller values correspond to *future* values of t_t (i.e., $t_t>18$).

Predicting Operational Quality

Similarly, the tradeoff between testing and quality can be analyzed by using equation (12), which is a useful measure of the *operational quality* of software because it measures the degree to which faults have been removed from the software (using the assumption that the faults that cause failures are removed), relative to predicted *maximum failures*. This type of quality is called *operational* (i.e., based on executing the software) to distinguish it from static quality (e.g., based on the complexity of the software). Using given values of *p* and equations (11) and (12) and letting $t=t_t$, $r(t_t)$ and *Q* are computed, respectively. The values of $r(t_t)$ are then used in equation (13) to compute t_t. Like equation (12), equation (13) has the asymptotic property of a great amount of testing required to achieve high levels of quality.

Making Reliability Decisions

In making the decision about how long to test, t_t, the reliability criteria and risk assessment approach can be applied. Table 1 is used to illustrate the process. For $t_t=18$ (when the last failure occurred on *OIA*), $r_c=1$, and $t_m=8$ days (.267 intervals), *remaining failures, RCM for remaining failures, time to next failure, RCM for time to next failure,* and *operational quality* are shown. These results indicate that *criterion 2* is satisfied but not *criterion 1* (i.e., *CRITICAL* with respect to *remaining failures*); also *operational quality* is low.

By looking at Table 1, it can be seen that if *remaining failures* r(18) are reduced by 1 from 4.76 to 3.76 (non-integer values are possible because the predictions are mean values), the predicted *time to next failure* that would be achieved is $T_F(18)=3.87$ intervals. These predictions satisfy *criterion 2* (i.e., $T_F(18)=3.87>t_m=.267$) but not *criterion 1* (i.e., $r(18)=4.76>r_c=1$). Note also in Table 1 that *fraction of remaining failures* $p=1-Q=.40$ at $r(18)=4.76$. Now, if testing is continued for a total time $t_t=52$ intervals, as shown in Table 1, and *remaining failures* are reduced from 4.76 to .60, the predicted *time to next 4.16 failure*s that would be achieved is 33.94 (34, rounded) intervals. This corresponds to $t_t=18+34=52$ intervals. That is, if testing is continued for an additional 34 intervals, starting at interval 18, another 4.16 failures would be expected. These predictions now satisfy *criterion 1* because $r(52)=.60<r_c=1$. Note also in Table 1 that *fraction of remaining failures* $p=1-Q=.05$ at $r(52)=.60$. Using the converse of the relationship, provides another perspective, where, if testing is continued for an additional $T_F=34$ intervals, starting at interval 18, the predicted *reduction in remaining failures* that would be achieved is 4.16 or r (52)=. 60.

Lastly, Figure 6 shows the *Deployment Decision*, relevant to the *Shuttle* (which could be the *Launch Decision* relative to the *Shuttle*), where *remaining failures* are plotted against *total test time* for *OIA*.

With these results in hand, the software manager can decide whether to deploy the software based on factors such as predicted *remaining failures*, as shown in Figure 6, along with considering other factors such as the trend in reported faults over time, inspection results, etc.. If testing were to continue until t_t=52, the predictions in Figure 6 and Table 1 would be obtained. These results show that *criterion 1* is now satisfied *(i.e., DESIRED*) and *operational quality* is high. Figure 6 also shows that at t_t=52, further increases would not result in a significant increase in reliability. Also note that at t_t=52 it is not feasible to make a prediction of T_F(52) because the predicted *remaining failures* is less than one.

Table 1. Reliability Criteria Assessment of OIA

					r_c=1			t_m=8 days	
t_t	a	ß	s^*	$r(t_t)$	RCM $r(t_t)$	s^*	$T_F(t_t)$	RCM $T_F(t_t)$	Q
18	.534	.061	9	4.76	3.76	9	3.87	-13.49	.60
52	.534	.061	9	.60	-.40	9	*	*	.95

* Cannot predict because predicted Remaining Failures is less than one.

<u>Summary of Predictions</u>

Table 2 shows a summary of remaining and maximum failure predictions compared with actual failure data, where available, for *OIA*, *OIB*, *OIC*, and *OID*. The purpose of this analysis is to validate the model for *Shuttle* applications. Because *actual* remaining and maximum failures are unknown, the assumption is used: that *remaining failures* are "zero" for those OI's (*B*, *C*, and *D*) that were executed for extremely long times (years) with no additional failure reports; correspondingly, for these OI's, the assumption is used that *maximum failures* equals total observed failures.

Table 2. Predicted Remaining and Maximum Failures versus Actuals

	t_t	s^*	a	ß	$r(t_t)$	Actual r	F(¥)	Actual F
OIA	18	9	.534	.061	4.76	?[A]	11.76	7[A]
OIB	20	1	1.69	.131	0.95	1[B]	12.95	13[B]
OIC	20	7	1.37	.126	1.87	2[C]	12.87	13[C]
OID	18	6	.738	.051	7.36	4[D]	17.36	14[D]

Time of last recorded failure:
A. No additional failures have been reported after 17.17 intervals for OIA.
B. The last recorded failure occurred at 63.67 intervals for OIB.
C. The last recorded failure occurred at 43.80 intervals for OIC.
D. The last recorded failure occurred at 65.03 intervals for OID.

Table 3 shows a summary of *total test time* and *time to next failure* predictions compared with actual execution time data, where available, for *OIA*, *OIB*, *OIC*, and *OID*.

Table 3. Predicted Total Test Time and Time to Next Failure versus Actuals

	s^*	$t_t(r=1)$	Actual t_t	t	s^*	$T_F(t)$	Actual T_F
OIA	9	43.59	?	18	9	3.9	?
OIB	1	*	63.67	20	0	*	43.67
OIC	7	24.98	27.07	20	5	4.2	7.63
OID	6	56.84	58.27	18	5	6.4	6.20

* Cannot predict because predicted Remaining Failures is less than one.
Additional Predictions for OID:
The following are additional predictions of total test time for OID that are not listed in Table 3: $t_t(r=2)=43.35$, Actual=45.17; $t_t(r=3)=35.47$, Actual=23.70.

Lessons Learned

Several important lessons have been learned from our experience of twenty years in developing and maintaining the PASS, which you could consider for adoption in your SRE process:

1) No one SRE process method is the "silver bullet" for achieving high reliability. Various methods, including formal inspections, failure modes analysis, verification and validation, testing, statistical process management, risk analysis, and reliability modeling and prediction must be integrated and applied.

2) The process must be continually improved and upgraded. For example, recent experiments with software metrics have demonstrated the potential of using metrics as early indicators of future reliability problems. This approach, combined with inspections, allows many reliability problems to be identified and resolved prior to testing.

3) The process must have feedback loops so that information about reliability problems discovered during inspection and testing is fed back not only to requirements analysis and design for the purpose of improving the reliability of future products but also to the requirements analysis, design, inspection and testing *processes* themselves. In other words the feedback is designed to improve not only the product but also the processes that produce the product.

4) Given the current state-of-the-practice in software reliability modeling and prediction, practitioners should *not* view reliability models as having the ability to make highly accurate predictions of future software reliability. Rather, software managers should interpret these predictions in two significant ways: a) providing increased confidence, when used as part of an integrated SRE process, that the software is safe to deploy; and 2) providing bounds on the reliability of the deployed software (e.g., high confidence that in operation the time to next failure will exceed the predicted value and the predicted value will safely exceed the mission duration).

References

[1] Recommended Practice for Software Reliability, R-013-1992, *American National Standards Institute/American Institute of Aeronautics and Astronautics*, 370 L'Enfant Promenade, SW, Washington, DC 20024, 1993.

[2] C. Billings, et al, "Journey to a Mature Software Process", *IBM Systems Journal*, Vol. 33, No. 1, 1994, pp. 46-61.

[3] E. W. Dijkstra, "Structured Programming", *Software Engineering Techniques*, eds. J. N. Buxton and B. Randell, NATO Scientific Affairs Division, Brussels 39, Belgium, April 1970 pp. 84-88.

[4] William H. Farr and Oliver D. Smith, *Statistical Modeling and Estimation of Reliability Functions for Software (SMERFS) Users Guide*, NAVSWC TR-84-373, Revision 3, Naval Surface Weapons Center, Revised September 1993.

[5] *IEEE Standard Glossary of Software Engineering Terminology*, IEEE Std 610.12.1990, The Institute of Electrical and Electronics Engineers, New York, New York, March 30, 1990.

[6] Ted Keller, Norman F. Schneidewind, and Patti A. Thornton "Predictions for Increasing Confidence in the Reliability of the Space Shuttle Flight Software", *Proceedings of the AIAA Computing in Aerospace 10*, San Antonio, TX, March 28, 1995, pp. 1-8.

[7] Norman F. Schneidewind, "Reliability Modeling for Safety Critical Software", *IEEE Transactions on Reliability*, Vol. 46, No.1, March 1997, pp.88-98.

[8] Norman F. Schneidewind, "Software Reliability Model with Optimal Selection of Failure Data", *IEEE Transactions on Software Engineering*, Vol. 19, No. 11, November 1993, pp. 1095-1104.

[9] Norman F. Schneidewind and T. W. Keller, "Application of Reliability Models to the Space Shuttle", *IEEE Software*, Vol. 9, No. 4, July 1992 pp. 28-33.

[10] Norman F. Schneidewind, "Analysis of Error Processes in Computer Software", *Proceedings of the International Conference on Reliable Software*, IEEE Computer Society, 21-23 April 1975, pp. 337-346.

[11] Elaine J. Weyuker, "Using the Consequences of Failures for Testing and Reliability Assessment", *Proceedings of the Third ACM SIGSOFT Symposium on the Foundations of Software Engineering*, Washington, D.C., October 10-13, 1995, pp. 81-91.

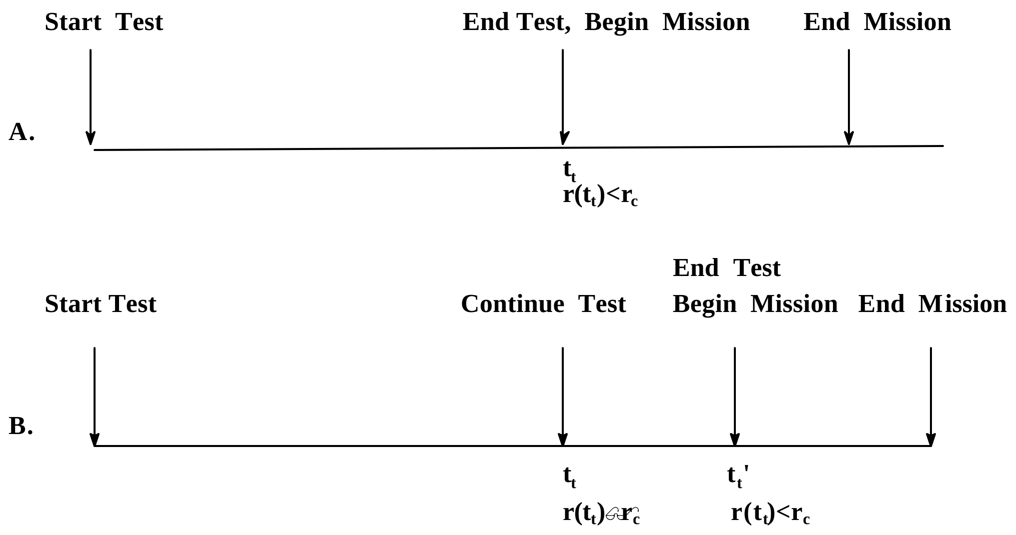

Figure 1. Remaining Failures Criterion Scenario

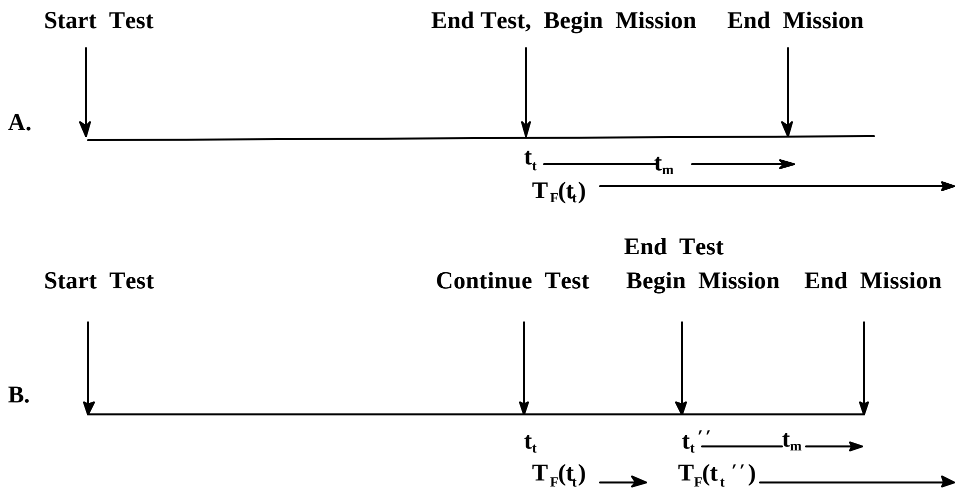

Figure 2. Time to Next Failure Criterion Scenario

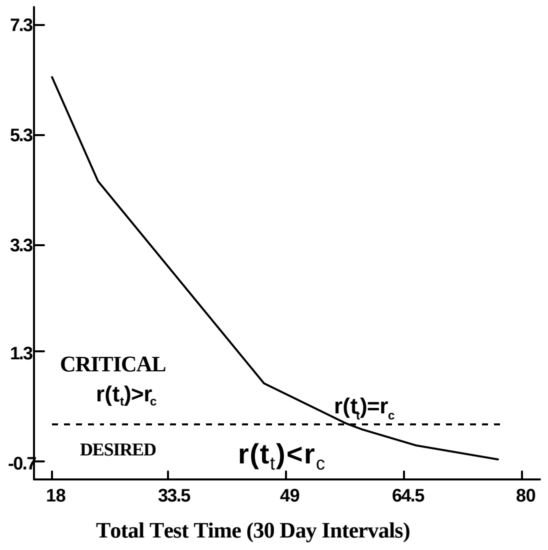

Figure 3. RCM for Remaining Failures, (r_c=1), OID

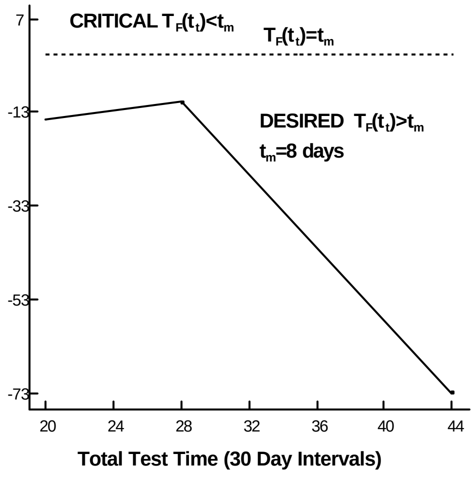

Figure 4. RCM for Time to Next Failure (t_m=8 days), OIC

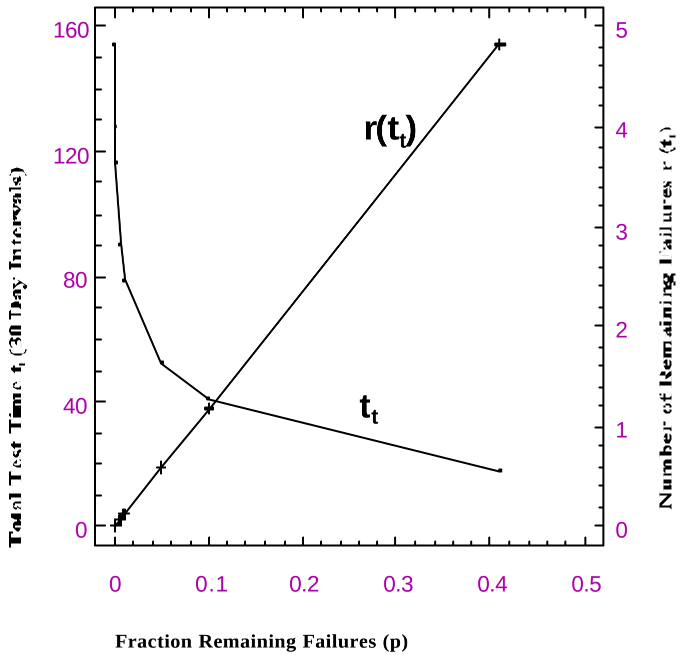

Figure 5. Total Test Time and Remaining Failures vs. Fraction Remaining Failures, OIA

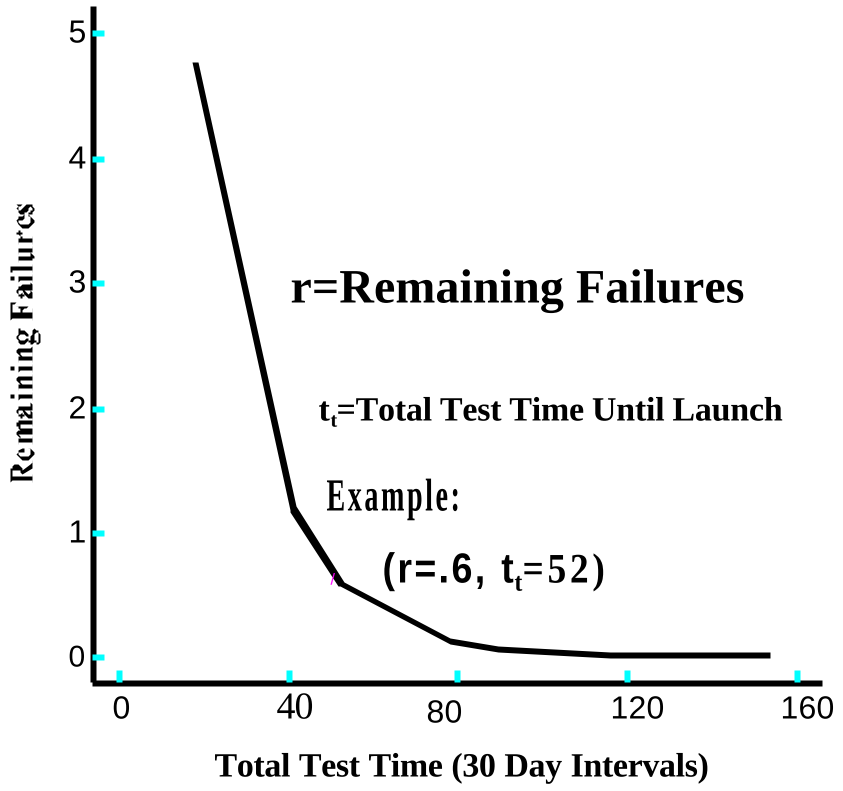

Figure 8. Launch Decision: Remaining Failures vs. Total Test Time, OIA